VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Autonomní řízení auta - optimalizace detekce dráhy

Autonomous Car Control - Track Detection Optimization

2018 Vojtěch Ihn

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Vojtěch Ihn**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Autonomní řízení auta - optimalizace detekce dráhy
Autonomous Car Control - Track Detection Optimization

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je optimalizovat způsob snímání jízdní dráhy pro model auta. Dráha se snímá řádkovou kamerou. Pro výstup kamery je potřeba navrhnout a vyzkoušet vhodné filtry pro redukci šumu. Ze snímků kamery detekovat trajektorii obrysových čar a stanovit odhad středu jízdní dráhy.

1. Seznamte se s metodami filtrování digitálního obrazu a detekce hran.
2. Implementujte vybrané filtry a otestujte je na reálných datech získaných z řádkové kamery při různých světelných podmínkách.
3. Implementujte vybrané algoritmy pro detekci hran a zohledněte délku historie dat z řádkové kamery.
4. Porovnejte dosažené výsledky a vyberte nejvhodnější kombinaci filtru, detekce hran a délku historie záznamu.
5. Implementujte vybranou kombinaci filtrů a detekce hran pro řídící mikropočítač modelu auta.
6. Otestujte a vyhodnoťte spolehlivost navrženého řešení při různých rychlostech a světelných podmínkách.

Seznam doporučené odborné literatury:

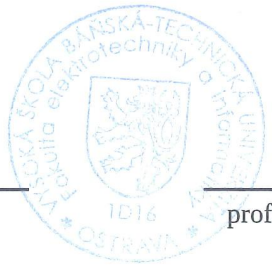
[1] R. C. Gonzalez and R. E. Woods, Digital Image Processing (third edition), Reading, Massachusetts: Addison-Wesley, 1992.
[2] Sojka E., Digitální zpracování a analýza obrazu, skripta VŠB
[3] Kinetis Design Studio IDE, http://www.nxp.com
[4] SDK 2.x pro mikropočítač K64F, http://www.nxp.com

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Petr Olivka, Ph.D.**

Datum zadání: 01.09.2017

Datum odevzdání: 30.04.2018



doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 30. dubna 2018 .....

Rád bych poděkoval svému vedoucímu práce Ing. Petru Olivkovi, Ph.D. za jeho pomoc a vytrvalost při řešení této práce.

Abstrakt

Tato práce popisuje ovládání modelu automobilu používaného na mezinárodní soutěži NXP Cup, konkrétně zpracování obrazu získaného z řádkové kamery a následnému hledání pozic okrajových čar dráhy. Jsou provedeny testy objektivů kamery a grafických filtrů, ze kterých je následně vybrána nejvhodnější kombinace. Tyto vybrané filtry jsou následně naimplementovány k použití na desce FRDM-K64F.

Klíčová slova: Filtrování obrazu, rozpoznávání čar, NXP Cup, FRDM-K64F

Abstract

This bachelor thesis describes work with a car model, which is used in an international competition called NXP Cup. The main focus is on processing of a data obtained from a line camera on the model using image processing techniques and detection of boundary lines on the track. Different camera objectives and image filters were tested, from which the most suitable ones were selected. The selected filters were then implemented on a FRDM-K64F developement platform.

Key Words: Image processing filters, line detection, NXP Cup, FRDM-K64F

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Cílem této bakalářské práce je vytvořit spolehlivé rozpoznávaní okrajových čar dráhy na modelu automobilu používaném na soutěži NXP Cup[4].

Na tuto práci navazuje další bakalářská práce, která model automobilu podle zjištěných čar ovládá. Kombinovaným výsledkem obou prací je tedy autonomně ovládaný model automobilu, který je schopen jezdit po předem definovaném typu dráhy.

Autonomní řízení automobilů je v dnešní době velmi populárním tématem a výrobci automobilů se předhánějí o prvenství v této oblasti. Jedná se ovšem o velmi komplexní oblast, kde musí být perfektně vyřešeny všechny možné situace, jelikož jakýkoliv přehlédnutý detail může skončit i smrtí cestujících.

Z tohoto důvodu je autonomní řízení částí populace stále vnímáno jako něco nepřijatelného, ovšem rychlost vývoje naznačuje, že během několika let dojde k výraznému rozšíření a potkávání autonomně řízených aut bude na denním pořádku.

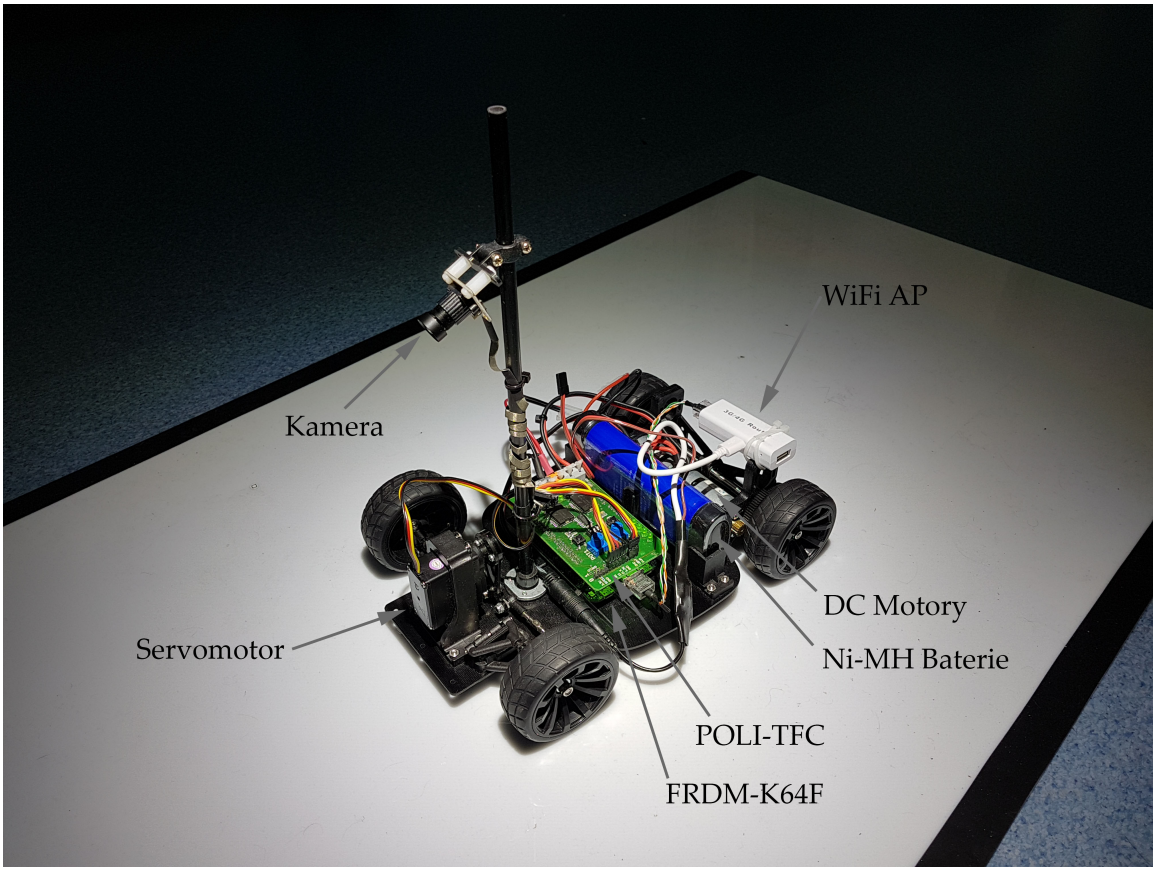
V této práci budou otestovány různé kombinace kamer a objektivů. Po zvolení té nejvhodnější přijde řada na testování filtrů sloužících k redukci šumu, detekci hran a prahování. Následně budou vybrané filtry naimplementovány k použití na mikropočítači použitém v modelu automobilu.

2 Model automobilu

V této práci je použit model automobilu od firmy Landzo nazvaný Alamak[1]. Tento model byl schválen firmou NXP k používání v soutěži NXP Cup[4] od roku 2018 a jedná se o výkonnější alternativu k modelu používanému v minulých ročnících, nazvanému Model-C.

K ovládání tohoto modelu slouží 2 motory řízené pomocí PWM, každý poháněj́ící jedno zadní kolo, a servomotor sloužící k natáčení předních kol. Ke snímání dráhy je použita řádková kamera umístěná v přední části modelu.

Jako řídící elektronika je použita vývojová deska FRDM-K64F[2] od firmy NXP spolu s modulem POLI-TFC zapojeným do GPIO pinů této desky. Do Ethernet portu řídící desky je zapojený WiFi Access Point, který slouží k odesílání dat z řídící desky na zařízení připojená k tomuto Access Pointu. Vše je napájeno NiMH baterií o napětí 7.2 V. Kompletní model je vyobrazen na obrázku 1.



Obrázek 1: Model automobilu

3 Řídící moduly a komunikace s nimi

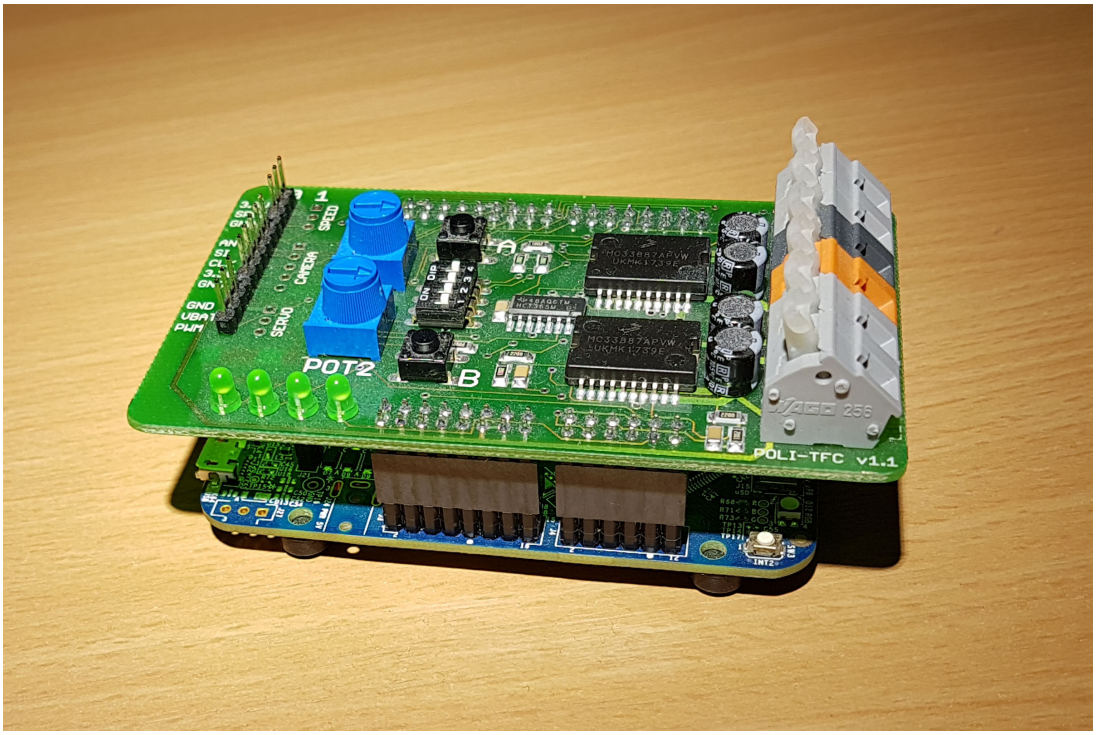
3.1 Vývojová deska FRDM-K64F

Jedná se o desku od společnosti NXP Semiconductors s mikrokontrolérem postaveným na jádru ARM Cortex-M4. Deska obsahuje 2 micro USB porty, z toho jeden slouží k programování desky, Ethernet port a slot pro microSD kartu.

3.2 Modul POLI-TFC

Jedná se o vlastní modul od vedoucího práce, který slouží k ovládání až dvou DC motorů pomocí H-můstků, dvou servomotorů a umožňuje připojení až dvou řádkových kamer. Modul vychází z oficiálního FRDM-TFC[3] firmy NXP, od kterého se liší pouze ve způsobu připojení pinů.

Prvky připojené k tomuto modulu lze ovládat dvěma způsoby. Veškeré ovládání může obstarávat pouze deska FRDM-K64F, nebo lze alternativně využít externího zařízení připojeného k USB nebo Ethernet portu této desky, které ovládání zajistí pomocí komunikačního protokolu popsanému v podkapitole 3.3.



Obrázek 2: Deska FRDM-K64F s modulem POLI-TFC

3.3 Komunikační protokol

Ke komunikaci s řídicí deskou přes sériovou linku, nebo jako v tomto případě, přes Ethernet, je možno využít třech různých datových struktur. Každá takováto struktura musí být zaobalena v datovém rámci zobrazeném v tabulce 1. Samotné datové struktury jsou zobrazeny ve výpisu 1.

Tabulka 1: Datový rámec

	STX	**length**	**CMD**	**data**	**ETX**
Počet bajtů	1	2	1	Dle použité struktury	1
Hodnota	0x02	0x00 - 0xFF	0x01 - 0x03	Data	0x03

```
struct tfc_data_s{
   uint32_t timestamp;
   uint16_t adc[ anLast ];
   uint8_t dip_sw;
   uint8_t push_sw;
   uint16_t image[TFC_CAMERA_LINE_LENGTH];
   uint32_t _padding;
};

struct tfc_setting_s{
   uint16_t servo_center[ 2 ];
   uint16_t servo_max_lr[ 2 ];
   uint16_t pwm_max;
   uint16_t _padding;
};

struct tfc_control_s{
   uint8_t leds;
   uint8_t pwm_onoff;
   uint8_t servo_onoff;
   uint8_t _padding1;
   int16_t pwm[ 2 ];
   int16_t servo_pos[ 2 ];
};
```

Výpis 1: Datové struktury

3.3.1 Datový rámec

V datovém rámci označují `STX` a `ETX` bajty začátek a konec samotného rámce, což je potřeba zejména při použití sériové komunikace, jelikož po sériové lince chodí data kontinuálně a bez těchto bajtů by od sebe nešly spolehlivě rozeznat jednotlivé rámce. V případě posílání dat pomocí UDP protokolu, jako v téhle práci, tyto bajty nejsou důležité, protože jsou data rozdělena do samostatných UDP paketů.

Bajt `CMD` slouží pouze k informaci o typu zasílané datové struktury a může nabývat třech hodnot:

- `CMD_DATA` (0x01) označuje datovou strukturu `tfc_data_s`,
- `CMD_SETTING` (0x02) označuje datovou strukturu `tfc_setting_s`,
- `CMD_CONTROL` (0x03) odpovídá datové struktuře `tfc_control_s`.

Položka `data` již obsahuje samotnou datovou strukturu naplněnou potřebnými daty, jejíž velikost v bajtech je zapsána v položce `length`. Tato hodnota je opět potřebná pro komunikaci po sériové lince.

3.3.2 Struktura `tfc_data_s`

Tato struktura slouží výhradně k získávání dat z řídící desky. Položka `timestamp` slouží jako pořadové číslo a dvě po sobě jdoucí datové struktury by na sebe měly navazovat právě touto hodnotou.

Položka `adc` je pole hodnot v rozmezí 0x0 ÷ 0xFFF získaných přímo z A/D převodníku a každý index lze považovat za hodnotu výčtu `tfc_andata_chnl_enum` a odpovídá určitému prvku:

- `anPOT_1` (index 0) a `anPOT_2` (index 1) odpovídají oběma potenciometrům,
- `anFB_A` (index 2) a `anFB_B` (index 3) jsou hodnoty feedbacku z DC motorů,
- `anBAT` (index 4) odpovídá napětí baterie.

Hodnota položky `dip_sw` odpovídá DIP přepínači a v `push_sw` jsou uloženy stavy obou tlačítek.

Položka `image` poté již obsahuje samotná data z kamery, konkrétně se jedná se o jeden řádek se šířkou 128 pixelů, kde každá hodnota je v rozmezí 0x0 ÷ 0xFFF jako hodnoty získané z A/D převodníku.

Položka `_padding` již neobsahuje žádná data a slouží spíše jako rezerva pro případné rozšíření této datové struktury.

3.3.3 Struktura `tfc_control_s`

Tuto strukturu lze využít dvěmi způsoby. Lze ji použít k ovládání servomotorů, DC motorů a LED diod, nebo k získávání hodnot z těchto prvků. V této práci je použita pouze k získávání hodnot, tudíž je posílána z řídící desky na ethernetové rozhraní, stejně jako struktura `tfc_data_s`.

Položka `leds` se váže ke čtyřem LED diodám na desce POLI-TFC, kde stav každé diody (zapnuto/vypnuto) odpovídá jednomu bitu ve spodní polovině bajtu.

Stav DC motorů je uložen v položce `pwm_onoff` a samotné hodnoty v rozmezí -1000 – 1000 se nachází v poli `pwm`, které obsahuje hodnoty pro oba motory. Totéž platí i pro servomotory a položky `servo_onoff` a `servo_pos`.

Položka `_padding1` opět slouží jako rezerva pro případné rozšíření, stejně jako u datové struktury `tfc_data_s`.

3.3.4 Struktura `tfc_setting_s`

Použití této datové struktury se liší od struktur předchozích. Slouží k počáteční kalibraci prvků desky a tudíž není neustále posílána, ať už z řídicí desky, nebo na ni. V případě této práce je vytvořena lokálně na řídící desce a nastavena na výchozí hodnoty, které lze upravovat metodami, které budou popsány v podkapitole 3.4.

Položka `servo_center` slouží k nastavení středové polohy servomotorů, od které se poté počítá rozsah ±1000. Položka `servo_max_lr` poté slouží k omezení tohoto rozsahu.

Pomocí položky `pwm_max` lze omezit maximální výkon motorů, z maximálního rozsahu ±1000 na nižší.

3.3.5 Použitá komunikace v této práci

Data jsou posílána pouze jednosměrně, a to z řídící desky na Ethernetové rozhraní pomocí UDP protokolu. V jednom paketu jsou vždy odeslány čtyři dvojice struktur `tfc_data_s` a `tfc_control_s` a za jednu vteřinu je odesláno 25 takovýchto paketů. Toto množství bylo zvoleno na základě maximální velikosti Ethernetového/WiFi rámce (1500 bajtů).

3.4 Ovládání funkčních prvků

K jednoduchému ovládaní všech prvků je vytvořena třída TFC se všemi potřebnými metodami, které zaobalují metody pracující přímo s HW. Metody lze rozdělit do následujících kategorií:

- Ovládání LED diod a přepínačů – metody `setLED()` a `setLEDs()` k nastavení jedné nebo více diod, `getDIPSwitch()` k získání hodnoty z DIP přepínače a `getPushButton` ke zjištění stavu tlačítek,

- čtení analogových hodnot – metoda `ReadADC()` k přečtení hodnot z A/D převodníku z prvků zmíněných u pole `adc` struktury `tfc_data_s`, `ReadPot_i()` a `ReadPot_f()` k získání hodnot z potenciometrů v rozsahu -0xFFF ÷ 0xFFF nebo -1,0 ÷ 1,0, `ReadFB_i()` k získání hodnot feedbacku z DC motorů v rozsahu 0x0 ÷ 0xFFF a `ReadFB_f()` k získání skutečného proudu protékajícího motory v ampérech a `ReadBatteryVoltage_i()` spolu s `ReadBatteryVoltage_f()` k získání napětí na baterii v rozsahu 0x0 – 0xFFF, resp. skutečného napětí ve voltech,

- čtení dat z kamer – metoda `ImageReady()` pro zjištění, zda je k dispozici nový obraz z kamery a `getImage()` pro získání samotného obrazu,

- ovládání motorů – metoda `MotorPWMOnOff()` zapíná a vypíná motory, pomocí metody `setPWMMax()` lze omezit maximální výkon motorů, metoda `setMotorPWM_i()` nastavuje výkon motorů v rozmezí -1000 ÷ 1000 a tyto nastavené hodnoty je možné zjistit pomocí metody `getMotorPWM_i()` a pomocí metody `setMotorPWM_f()` lze také nastavit výkon motorů, ovšem v rozmezí -1.0 ÷ 1.0,

- ovládaní servomotorů – jsou použity podobné metody, jako u ovládání motorů, tzn. `ServoOnOff()` k zapnutí/vypnutí, `setServoCalibration()` ke kalibraci a `setServo_i()`, `getServo_i()` spolu s `setServo_f()` k nastavování a získávání hodnoty natočení.

Třída obsahuje lokální proměnnou typu `tfc_setting_s`, do které jsou po vytvoření třídy metodou `InitAll()` volanou v konstruktoru uloženy výchozí hodnoty kalibrace. Tyto hodnoty je poté možné měnit již zmíněnými metodami `setPWMMax()` a `setServoCalibration()` a jsou používány k modifikaci hodnot, které jsou předávány metodám pracujícím přímo s HW.

Deklarace třídy se nachází v souboru `tfc.h`, který byl v rámci této práce zdokumentován pomocí nástroje Doxygen[5]. Smotný obsah souboru lze nalézt v příloze A, vygenerovanou dokumentaci následně na přiloženém CD.

4 Dráha

Použitá dráha může být poskládaná z několika různých částí vyrobených z bílého plexiskla, které lze rozdělit do tří kategorií:

- Části se dvěmi viditelnými okraji – jedná se o rovinku, kopec a zvlněnou rovinku. Šířka těchto částí je vždy 606 mm včetně okrajových čar o tloušťce 25.4 mm.
- Zatáčky – při průjezdu zatáčkou jde na řádkové kameře vždy vidět jenom jeden, vnější okraj. Poloměr vnitřního okraje je 303 mm, u vnějšího okraje se jedná o 909 mm a každá zatáčka má 90 stupňů.
- Části bez viditelných okrajů – v dráze se mohou nacházet křižovatky, kterými prochází dráha ve dvou na sebe kolmých směrech a proto při průjezdu této křižovatky nejsou na kameře vidět žádné okraje.

Obrázek 3: Dráha obsahující všechny typy úseků

5 Aplikace pracující s výstupními daty

K usnadnění práce s daty z automobilu a následnému testování byly vytvořeny dvě aplikace v jazyce C++.

První aplikace přijímá na síťovém rozhraní UDP pakety zmíněné v podkapitole 3.3.5, a následně přijatá data vykresluje. Tímto umožňuje v reálném čase sledovat obraz z kamery a informace, jako aktuální natočení servomotoru nebo výkon pohánějících motorů. Dále tato aplikace umožňuje přijatá data uložit do binárních souborů.

Tyto soubory je poté možné načíst v druhé aplikaci, která následně uložená data odesíla se specifickými časovými rozestupy a tímto simuluje chování samotného automobilu. Díky tomuto je možné pomocí první aplikace vytvořit soubor se záznamem, který je poté možné kdykoliv znovu odeslat, například pro otestování jiné kombinace filtrů.

Odesílání a přijímání dat bylo vyřešeno pomocí socketů[6]. Prohlížeč dat naslouchá na určitém portu a čeká, dokud neobdrží UDP paket zmíněný v předchozí kapitole. Při jeho obdržení vykreslí přijatá data a toto se opakuje s každým nově přijatým paketem. Při odesílání je kromě portu nastavena i cílová IP adresa, na kterou jsou pakety posílány.

Přímo pro toto použití byly vytvořeny 2 třídy, jedna sloužící k přijímání a druhá k odesílání dat. Implementace socketů se ovšem u různých operačních systémů liší v malých detailech, a proto byl vytvořen hlavičkový soubor uvedený níže, který může být naimplementován ve více souborech, kde každý bude odpovídat jinému operačnímu systému.

Konkrétní implementace byly vytvořeny pro operační systémy Windows a Linux. Rozdíly v těchto implementacích jsou minimální, například Windows používá pro uložení socketu datový typ `SOCKET` a Linux jednoduchý `int`, ovšem i tyto minimální rozdíly vedou k nepřenositelnosti mezi těmito systémy.

```
class Socket {
protected:
   Socket() {};
   SOCKET s;
public:
   void close();
};

class SocketRx : public Socket {
public:
   SocketRx() {};
   int listen(int port);
   int recieveData(char* dataPtr, int size);
};
```

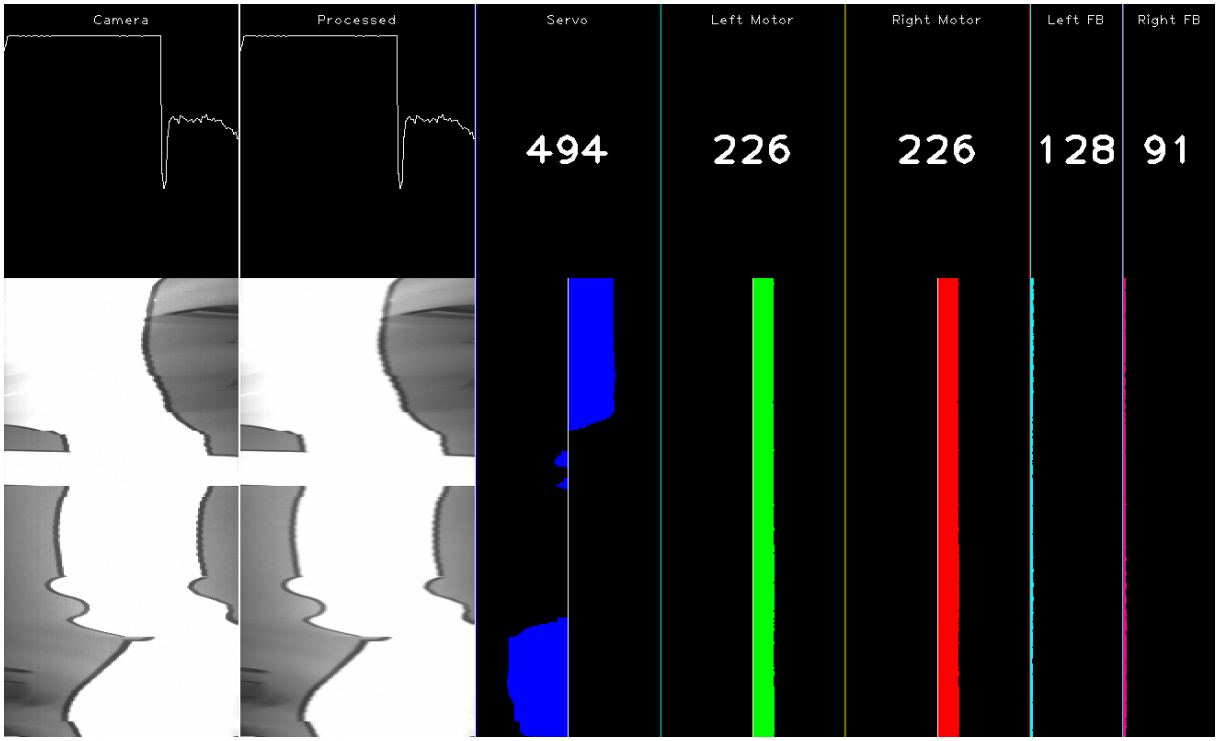
```
class SocketTx : public Socket {
private:
   sockaddr_in server;
public:
   SocketTx() {};
   int init(std::string ip, int port);
   int sendData(char* dataPtr, int size);
};
```

Výpis 2: Hlavičkový soubor pro sockety

5.1 Prohlížeč dat

K vytvoření této aplikace byla použita knihovna OpenCV[7]. Aplikace umožňuje přijímat data pomocí socketů zmíněných na začátku kapitoly 5, tato data vykreslit na obrazovku a případně je uložit do souboru.

Hlavní okno aplikace se skládá ze sloupců, které lze libovolně přidávat a odebírat. Každý z těchto sloupců slouží k vizualizaci jedné položky přijatých dat. Aplikace ve výchozím nastavení je vyobrazena na obrázku 4.



Obrázek 4: Vzhled prohlížeče dat

Aplikace v tomto rozložení obsahuje sloupce odpovídající následujícím prvkům:

- Kamera – první sloupec zobrazuje data přijatá přímo z kamery a druhý sloupec obsahuje obraz s použitými filtry,
- servomotor – třetí sloupec vizualizuje natočení servomotoru v rozsahu -1000 ÷ 1000,
- motory – Čtvrtý a pátý sloupec zobrazují PWM (výkon) obou motorů, opět v rozsahu -1000 ÷ 1000,
- feedback z motorů – šestý a sedmý sloupec obsahují feedback, což je proud procházející H-můstky, který je přímo úměrný proudu protékajícímu motory, získaný z A/D převodníku v rozsahu 0x0 ÷ 0xFFF.

Ve vrchní části je zobrazen detail každého sloupce, který odpovídá nejnovějšímu záznamu. V případě kamery se jedná o graf znázorňující hodnotu každého pixelu v rozmezí 0 ÷ 255 a u zbylých sloupců je pouze vypsána aktuální hodnota.

Pod tímto detailním náhledem je oblast obsahující historii přijatých dat. Při obdržení nového paketu zmíněného v podkapitole 3.3.5 je celá tato oblast posunuta dolů a nově přijaté záznamy jsou vykresleny na první řádky této oblasti.

```
void Chart::RenderColumn(size_t columnIndex, size_t y, uint16_t *values){
   const auto column = columns[columnIndex];
   for (int x = 0; x < column.Width; x++){
      short value = (values[x / CAMERA_SCALE] / 16) & 0xff;
      this->image.at<Vec3b>(y, column.Start + x) = Vec3b(value, value,
          value);
   }
   this->image.at<Vec3b>(y, column.Start) = column.Color;
   this->image.at<Vec3b>(y, column.End) = column.Color;
}
```

Výpis 3: Příklad vykreslení řádku z kamery

Přijatá data jsou v aplikaci neustále ukládána a v případě zadání názvu souboru, jako parametru při spuštění, jsou po ukončení aplikace uložena do binárního souboru, který je poté možné načíst a odeslat aplikací zmíněné podkapitole 5.2.

Filtrování obrazu může obstarávat buď knihovna OpenCV, nebo vlastní třída s vybranými filtry, která bude popsána v podkapitole 8.2.

5.2 Odesílání dat

Jedná se o velmi jednoduchou aplikaci, která načte uložený binární soubor do pole a poté odesílá pomocí třídy `Socket`, zmíněné na začátku kapitoly 5, struktury popsané v podkapitole 3.3.5 v časových rozestupech odpovídajících frekvenci odesílání přímo z řídící desky automobilu, tj. 25 struktur za vteřinu.

6 Výběr kamery a objektivu

K dispozici byly dvě kamery. Kamera č. 1 od firmy Freescale, která byla používána již na starém modelu automobilu, je vyobrazená na obrázku 5. Kamera č. 2 zobrazená na obrázku 6 je od firmy Landzo a byla součástí modelu Alamak[1]. Součástí obou kamer je řádkový CCD čip o rozlišení 1x128 pixelů.

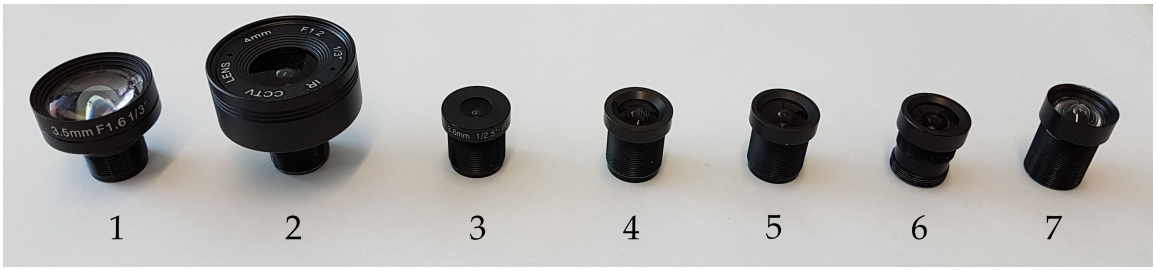
(a) Pohled zepředu (b) Pohled zezadu

Obrázek 5: Kamera č. 1 s objektivem č. 1

(a) Pohled zepředu (b) Pohled zezadu

Obrázek 6: Kamera č. 2 s objektivem č. 7

Tyto kamery jsou osazeny držákem objektivu se závitem M12, na který jdou našroubovat různé objektivy. K testování bylo k dispozici přibližně 10 objektivů, ze kterých byly některé hned zavrženy, buď z důvodu velmi malého zorného úhlu, velkého zkreslení nebo malé světelnosti. Ve výsledku bylo otestováno pouze 7 objektivů, které lze nalézt na obrázku 7.



Obrázek 7: Testované objektivy

Obě kamery se všemi objektivy byly testovány za stejných podmínek při umělém osvětlení. Pro testování byl speciálně vytvořen kus kalibrační dráhy, který je ze stejného materiálu, jako používaná dráha, oproti které ale obsahuje větší množství čar, viz. obrázek 8.

Obrázek 8: Úsek pro testovaní kamer a objektivů

Středová čára slouží k orientaci a horizontální čára k měření vzdálenosti od kamery ke snímané oblasti. Postranní čáry jsou od sebe vzdáleny vždy 10 cm a jsou určeny k otestování šířky záběru a případného zkreslení.

V tabulce 2 jsou výsledky měření jednotlivých objektivů s kamerou č. 1, tabulka 3 poté obsahuje výsledky při použití kamery č. 2. Tabulky obsahují hodnoty:

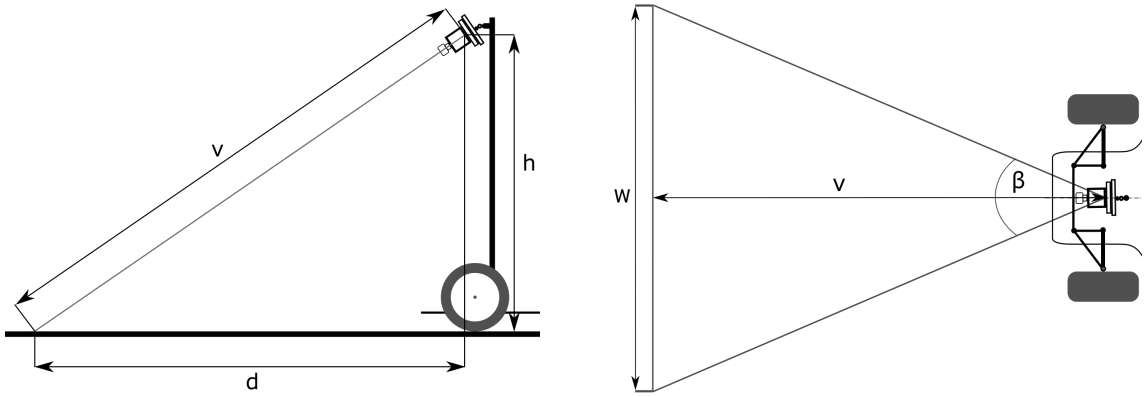
- Levý a pravý okraj – každý objektiv nemusí osvětlit celou šířku CCD čipu, a proto jsou uvedeny tyto hodnoty. Jedná se o pozice pixelů $0 \div 127$, od kterých už kamera zaznamenává velmi tmavý, nebo dokonce žádný obraz.

- Úhel záběru – tento úhel byl vypočítán ze známých hodnot, kterými jsou výška kamery $h = 275\ mm$, vzdálenost spodní části kamery od pozice snímané kamerou $d = 400\ mm$ (viz. obrázek 9).

  Z těchto hodnot byla pomocí Pythagorovy věty vypočítána vzdálenost kamery od snímané pozice $v = \sqrt{h^2 + d^2} = 485\ mm$.

Dále bylo využito známé šířky dráhy, která je 606 mm. Tato šířka odpovídá na každém záběru určitému počtu pixelů, a díky tomu byla vypočítána reálná šířka záběru w odpovídající celkovému počtu pixelů mezi levým a pravým okrajem.

Následně byl spočítán úhel $\beta = arctg(w/2v)$ a z něj celkový úhel záběru, který se rovná 2β.



(a) Pohled zboku

(b) Pohled zhora

Obrázek 9: Trojúhelníky použité k výpočtu úhlu

- Bílá a Černá – jedná se o hodnoty v rozsahu 0 – 255, které odpovídají průměrným hodnotám bílé barvy dráhy a černé barvy čar, které byly nasnímány konkrétní kombinací kamery a objektivu.

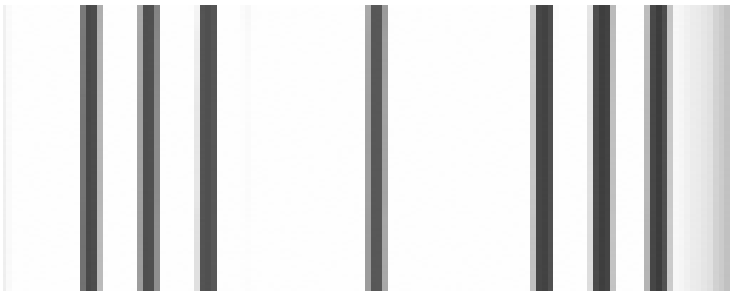
Tabulka 2: Porovnání objektivů s kamerou č. 1

Objektiv	**Levý okraj**	**Pravý okraj**	**Úhel záběru**	**Bílá**	**Černá**
1	15	106	79°	255	48
2	14	107	81°	255	40
3	14	110	79°	140	28
4	15	103	79°	120	28
5	18	105	82°	130	31
6	19	101	87°	190	39
7	4	123	87°	95	27

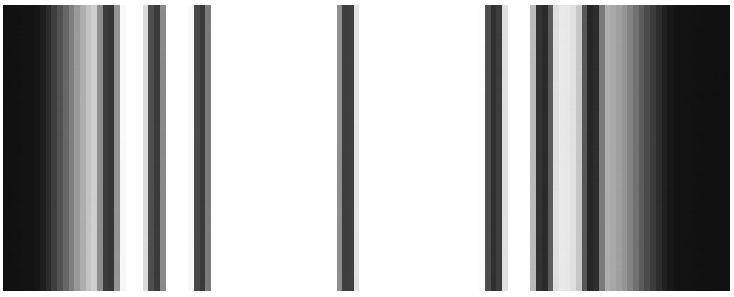
Tabulka 3: Porovnání objektivů s kamerou č. 2

Objektiv	Levý okraj	Pravý okraj	Úhel záběru	Bílá	Černá
1	11	116	84°	254	130
2	11	115	81°	254	210
3	14	113	78°	254	90
4	12	112	84°	254	100
5	15	112	85°	254	125
6	19	108	88°	254	120
7	4	123	84°	255	67

Z výsledků je patrné, že kamera č. 2 oproti kameře č. 1 velmi zvyšuje jas obrazu. Z tohoto důvodu jsou objektivy č. 1 a 2, které mají největší světelnost a podávají nejlepší výsledky s kamerou č. 1, na kameře č. 2 téměř nepoužitelné, jelikož rozdíl mezi bílou a černou barvou je minimální (viz. obrázek 13). Objektiv č. 7 využívá téměř celou šířku snímacího čipu a v kombinaci s kamerou č. 2 podává pravděpodobně nejlepší výsledek, a proto byla tato kombinace, jejíž výstup je zobrazen na obrázku 10, zvolena k dalšímu používaní. Dalšími dobře použitelnými variantami jsou kamera č. 1 v kombinaci s objektivy č. 1 (obrázek 11) nebo 2. Veškeré testovací snímky lze nalézt v příloze B.

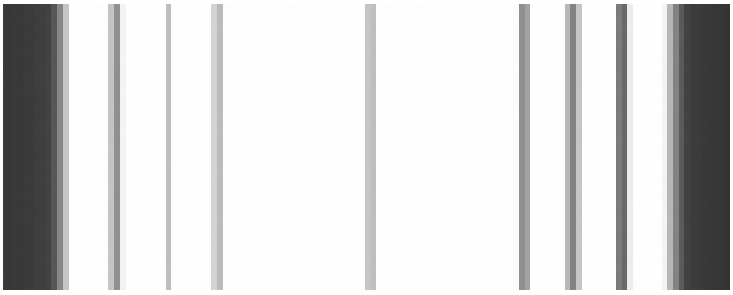


Obrázek 10: Zvolená varianta – kamera č. 2 a objektiv č. 7



Obrázek 11: Vhodná alternativa – kamera č. 1 a objektiv č. 1

Obrázek 12: Příklad příliš tmavého záznamu – kamera č. 1 a objektiv č. 4



Obrázek 13: Příklad příliš světlého záznamu – kamera č. 2 a objektiv č. 2

7 Zpracování obrazu

Na získaný obraz z kamery je nutné použít určitou kombinaci filtrů[9] na redukci šumu, detekci hran a prahování tak, aby po použití těchto filtrů bylo možné jednoduše zjistit pozici okrajových čar dráhy. K testování byly použity aplikace z kapitoly 5, pomocí kterých byly pořízeny záznamy v různých světelných podmínkách, na které následně byly pomocí knihovny OpenCV aplikovány filtry. Bylo potřeba vybrat filtry, které budou spolehlivé a ne příliš výpočetně náročné, aby mohly být bez problému použity i přímo na mikropočítači desky FRDM-K64F. K testování byly použity metody knihovny OpenCV s naimplementovanými filtry.

7.1 Konvoluce

Velké množství filtrů používá operátoru konvoluce[8]. Jedná se o binární operátor označený $*$, který ze dvou vstupních funkcí, konvolvované funkce $f(x)$ a konvolučního jádra $k(x)$, vytvoří novou funkci $(f * k)(x)$. V případě zpracování obrazu je funkcí $f(x)$ vstupní obraz s určitým rozlišením, a jedná se tedy o diskrétní funkci (funkce má definiční obor na $\mathbb{Z}$). Dále může konvoluce pracovat i se spojitými funkcemi, které mají definiční obor na $\mathbb{R}$, ovšem tato varianta zde není použita.

Při zpracování obrazu je tedy funkcí $f(x) \in \mathbb{Z}^d \mapsto \mathbb{R}$ (kde $d = 1$ odpovídá 1D obrazu a $d = 2$ 2D obrazu) samotný obraz tvořený pixely, kde x je souřadnice pixelu a funkce $f(x)$ vrací hodnotu pixelu na této pozici.

Konvoluční jádro $k_n(x)$ reprezentuje matice $1 \times n$ u 1D a $n \times n$ u 2D obrazu, kde n musí být vždy liché číslo.

Konvoluce pro 1D obraz je tedy definována následovně:

$$(f * k_n)(x) \coloneqq \sum_{i}^{\{-n,\ldots,n\}} f(x-i)k_n(i), \tag{1}$$

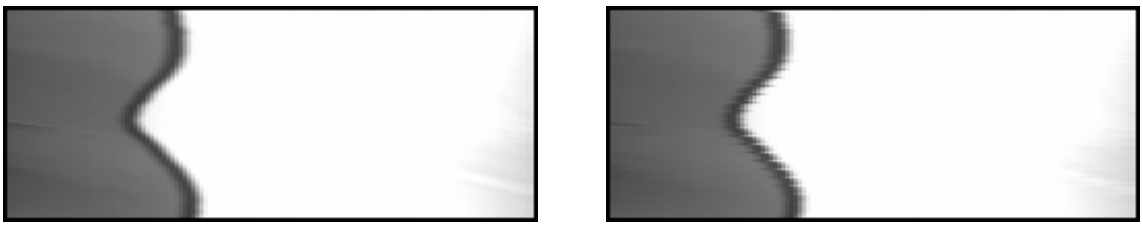
kde:

$(f * k_n)(x)$	je	nová hodnota pixelu x,
$f(x)$		hodnota pixelu x ve zdrojovém obrazu $f(x)$,
$k_n(i)$		hodnota jádra na pozici i,
n		velikost jádra.

Při konvoluci obrazu $f(x)$ jádrem $k_n(x)$ tedy vznikne nový obraz $(f*k)(x)$, kde nová hodnota pixelu na pozici x odpovídá váženému součtu pixelů v okolí n. Dá se tedy říct, že jádro $k_n(x)$ je jakási „tabulka“, jejíž střed $(i = 0)$ je položen na každý pixel a tabulka je následně převrácena podle středu. Poté jsou všechny pixely překryté touto tabulkou vynásobeny odpovídajícím koeficientem v tabulce a sečteny, čímž se získá nová hodnota pixelu.

7.2 Velikost filtrovaného obrazu

Při filtrování je možnost výběru dvou variant: filtrování vždy samostatného přijatého řádku, tzn. filtrování 1D obrazu, nebo přijatého řádku spolu s již uloženými řádky spojenými do 2D obrazu. Při testování se osvědčilo filtrování pouze jednoho řádku, které je méně náročné a lze u něj vypozorovat kvalitnější výsledky. Důvodem neefektivity filtrování na 2D obrazu je to, že polohu čar je potřeba zjistit z nejnovějšího řádku, který je vždy úplně nahoře, a pixely v tomto řádku plně nevyužijí 2D konvolučního jádra, jelikož toto jádro počítá i s pixely nad právě filtrovaným pixelem, které u tohoto řádku chybí. Bylo tedy zvoleno používat filtry vždy jenom na jednom řádku.



(a) 1D obraz (b) 2D obraz

Obrázek 14: Porovnání filtrování pomocí průměrování na 1D a 2D obrazu

7.3 Redukce šumu

Obraz z kamery nemusí být vždy bezchybný a může obsahovat nežádané artefakty, které mohou být způsobeny například nečistotami na objektivu nebo nepříliš ideálními světelnými podmínkami. Tento šum může způsobovat potíže u následného zpracování obrazu, jako například zde u detekce čar, a proto existují filtry, které se snaží zredukovat tento šum.

7.3.1 Mediánový filtr

Mediánový filtr nepracuje přímo na principu konvoluce zmíněném v podkapitole 7.1, ovšem i zde je používaná matice podobná konvolučnímu jádru. U každého pixelu jsou do pole uloženy všechny okolní pixely, do kterých zasahuje právě tato matice a toto pole je následně setříděno. Poté je filtrovanému pixelu přiřazena nová hodnota, která odpovída mediánu tohoto pole.

Tento filtr je vhodný pro úplné odstranění velmi malých artefaktů v obraze. Ovšem při zvolení velké hodnoty n u matice může filtr začít odstraňovat i hrany, které by měly být zachovány. Na obrázku 15 je simulace mrtvého pixelu na snímači kamery, který může být tímto filtrem úplně odstraněn. Při nastavení hodnoty $n = 5$ ale již začíná filtr odstraňovat i okraje samotné dráhy. K testování byla použita metoda `medianBlur()` knihovny OpenCV.

(a) Originální obraz (b) $n = 3$ (c) $n = 5$

Obrázek 15: Mediánový filtr s různými hodnotami n

7.3.2 Průměrování

Průměrování je filtr využívající konvoluci a konkrétně se jedná o jednoduchý případ, kde výsledná hodnota pixelu odpovídá aritmetickému průměru všech pixelů, do kterých zasahuje konvoluční jádro. Všechny prvky konvolučního jádra tedy mají stejnou hodnotu a jejich součet se rovná jedné. 1D konvoluční jádro s $n = 3$ tedy vypadá následovně:

$$\begin{bmatrix} 1/3 & 1/3 & 1/3 \end{bmatrix}. \tag{2}$$

Výhodou tohoto filtru je velmi nízká náročnost. Filtr je schopný odstranit ostré přechody způsobené například špatným osvětlením, může ovšem i znesnadnit rozpoznávání důležitých hran. K testování byla použita metoda `blur()` knihovny OpenCV.

(a) Originální obraz (b) $n = 3$ (c) $n = 5$

Obrázek 16: Průměrování s různými hodnotami n

7.3.3 Gaussův filtr

Jedná se o rozšíření obyčejného průměrování, kde každá hodnota v konvolučním jádru není stejná a hodnoty odpovídají distribuci Gaussovy křivky, která je pro 1D definována následovně:

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}}. \tag{3}$$

Ve výše zmíněné rovnici chybí parametr μ, který slouží k posunu po ose x. Tento posun je ale u filtrování obrazu nežádoucí, a proto se předpokládá, že se rovná 0. Tvar křivky tedy plně závisí na směrodatné odchylce σ. Gaussova křivka se ovšem šíří do nekonečna, a proto je potřeba hodnoty v konvolučním jádře s omezenou velikostí n lehce upravit tak, aby se jejich součet rovnal 1. 1D konvoluční jádro s $n = 5$ a $\sigma = 1$ tedy může vypadat takto:

$$\begin{bmatrix} 0.06136 & 0.24477 & 0.38774 & 0.24477 & 0.06136 \end{bmatrix}. \tag{4}$$

Výhodou oproti průměrování je lepší zachovávaní hran, jelikož pixely blíž ke středovému mají větší váhu, než pixely nacházející se dál. Nevýhodou může být vyšší výpočetní náročnost, pokud se nepoužívá statické jádro konvoluce a musí se počítat podle předaného parametru σ. K testování byla použita metoda `GaussianBlur()` knihovny OpenCV.

(a) Originální obraz (b) Průměrování s $n = 3$ (c) Gaussův filtr s $n = 5$

Obrázek 17: Porovnání Gaussova filtru a průměrování

7.3.4 Bilaterální filtr

Jedná se o komplexní filtr, který kombinuje výhody průměrování se schopností zachovávat hrany. Toho je dosaženo díky tomu, že filtr nepřiřazuje okolním pixelům váhu jenom podle jejich vzdálenosti od středového pixelu, ale bere v úvahu i jejich barvu a intezitu.

Tento filtr je náročnější, než většina jiných filtrů a jeho použití v této práci ztrácí smysl, jelikož na 1D obrazu nepodává příliš rozdílné výsledky oproti filtrům zmíněným výše. K testování byla použita metoda `bilateralFilter()` knihovny OpenCV.

(a) Originální obraz (b) Gaussův filtr s $n = 3$ a $\sigma = 1$ (c) Bilaterální filtr

Obrázek 18: Porovnání bilaterálního a Gaussova filtru

7.4 Detekce hran

Hrana v obrazu je definována jako prudká změna jasu sousedících pixelů. Takovéto hrany lze detekovat ve dvou, případně čtyřech směrech a často je použitá konvoluce.

7.4.1 Sobelův operátor

Sobelův operátor používá právě konvoluci, a pro každý ze 4 směrů je definováno specifické jádro. Jelikož jsou zde filtry používány pouze na 1D obrazu, je možné hledat hrany pouze ve směru osy x, a to pomocí následujícího jádra:

$$\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}. \tag{5}$$

Nová hodnota pixelu tedy odpovídá rozdílu hodnot jeho sousedících pixelů. Výsledkem můžou být i záporné hodnoty, které označují nárůst jasu v opačném směru. Často ovšem stačí znát pouze pozici hran, u kterých směr nárůstu jasu není důležitý, a proto se ukládají absolutní hodnoty získaných výsledků.

Rozpoznávání hran je velmi citlivé na šum, a proto je velmi vhodné před ním použít filtry na redukci šumu, případně ještě použít prahování na obraz s hranami. Ani poté ovšem není zaručeno, že budou vidět jen požadované hrany, a ne žádný nechtěný šum. K testování byla použita metoda `Sobel()` knihovny OpenCV.

(a) Originální obraz (b) Bez předchozích filtrů (c) S předchozím Gauss. filtrem

Obrázek 19: Porovnání Sobelova operátoru bez předchozích filtrů a s Gaussovým filtrem

7.4.2 Morfologický gradient

Morfologický gradient[10] je rozdílem dvou jednodušších operací: dilatace a eroze. Obě tyto operace pracují s maticí nazvanou strukturovací element, která je používána podobně jako jádro u konvoluce. Zde u 1D obrazu lze použít například matici:

$$\begin{bmatrix} 1 & 1 & 1 \end{bmatrix}. \tag{6}$$

Dilatace slouží ke hledání lokálního maxima. Jedná se o velmi podobnou operaci jako u mediánového filtru popsaného v podkapitole 7.3.1, ovšem místo mediánu se jako nová hodnota přiřadí ta maximální.

Eroze je přesným opakem dilatace, kde se místo lokálního maxima se ukládá lokální minimum.

Pokud se provedou obě operace, a výsledek eroze se odečte od výsledku dilatace, vznikne obraz se zvýrazněnými hranami, podobně jako u Sobelova operátoru. K testování byla použita metoda `morphologyEx()` knihovny OpenCV.

(a) Originální obraz (b) Bez předchozích filtrů (c) S předchozím Gauss. filtrem

Obrázek 20: Porovnání Morfologického gradiendu bez předchozích filtrů a s Gaussovým filtrem

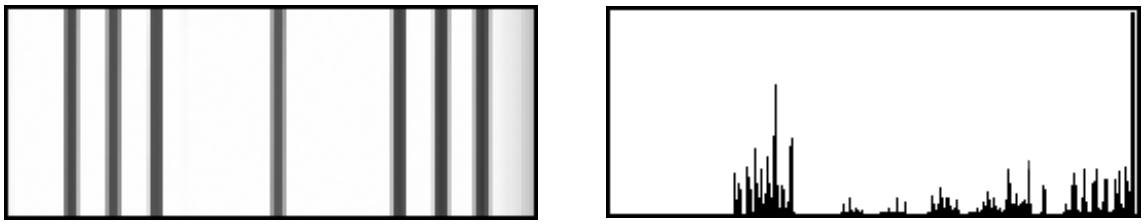
7.5 Prahování

Pomocí prahování se převádí černobílý obrázek na binární, skládající se pouze ze dvou hodnot. K tomu je nutno určit hodnotu prahu, se kterou se následně porovná každý pixel a pokud je jeho hodnota nižší, než práh, je nastaven na 0, jinak je nastaven na 1 nebo 255.

Nejdůležitější je správně nastavit právě hodnotu prahu. Ta může být nastavena buď ručně, což není ideální, nebo může být zjištěna pomocí algoritmu, který jeho hodnotu nastaví podle obrazu a například jeho histogramu.

7.5.1 Otsu prahování[11]

Tato metoda slouží k automatickému získání hodnoty prahu a pracuje s histogramem obrazu.



(a) Obraz z kamery (b) Histogram obrazu

Obrázek 21: Obraz a jeho histogram

Obraz je podle definice této metody rozdělen na pozadí a popředí a cílem je najít takovou hodnotu prahu T, při kterém je vnitřní rozptyl pozadí a popředí co nejnižší a mezi-rozptyl co nejvyšší.

Algoritmus prochází histogram obrazu, a zkouší každou hodnotu použít jako práh T. Pro každý práh T jsou nejdříve vypočítány váhy popředí a pozadí:

$$W_b = \sum_{i=0}^{T-1} p(i), \tag{7}$$

$$W_f = \sum_{i=T}^{L-1} p(i), \tag{8}$$

kde:

W_b	je	váha pozadí x,
W_f		váha popředí,
$p(i)$		poměr počtu pixelů s intezitou i k celkovému počtu pixelů v obraze,
T		hodnota práhu,
L		délka histogramu.

Následně jsou spočítány průměrné intenzity popředí a pozadí:

$$\mu_b = \sum_{i=0}^{T-1} i\frac{p(i)}{W_b}, \tag{9}$$

$$\mu_f = \sum_{i=T}^{L-1} i\frac{p(i)}{W_f}, \tag{10}$$

kde:

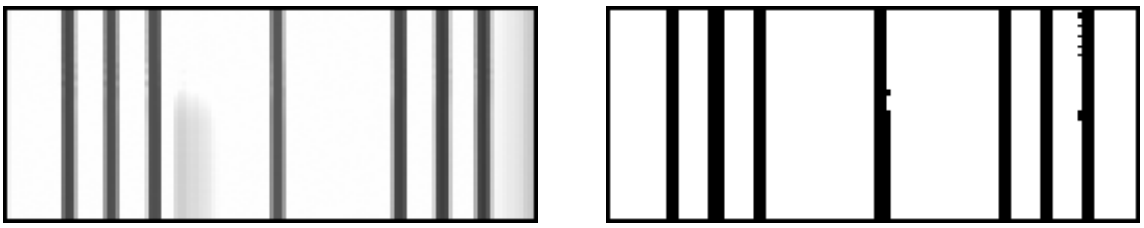
μ_b je průměrná intenzita pozadí x,
μ_f průměrná intenzita popředí.

Jako poslední krok je spočítán mezi-rozptyl pozadí a popředí, jehož výpočet je jednodušší, než výpočet vnitřního rozptylu:

$$\sigma_B^2 = W_b W_f(\mu_b - \mu_f)^2 \tag{11}$$

Výsledný práh T nakonec odpovídá hodnotě, při které byl mezi-rozptyl σ_B^2 nejvyšší.

Tato metoda prahování nemusí být vždy nejideálnější, ovšem v případě této práce se ověděila jako velmi spolehlivá. Nejlepší výsledky jsou dosaženy na obrazech, kde jsou na histogramu dvě výrazné, okrajové oblasti, což právě zde zastupuje bílá trať a černé čáry na okrajích. K testování byla použita metoda `threshold()` knihovny OpenCV.

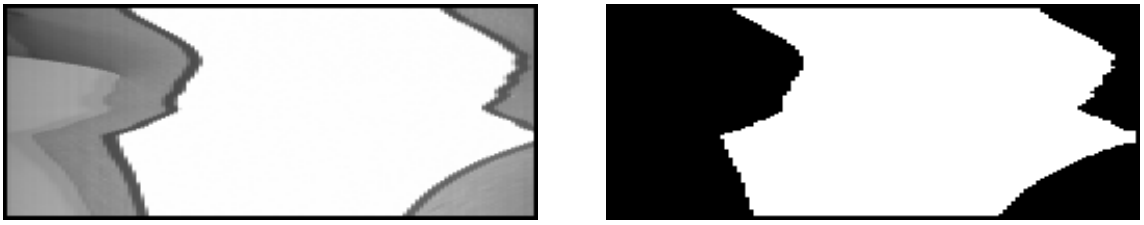


(a) Originální obraz (b) Použité prahování

Obrázek 22: Otsu prahování

7.6 Zvolené filtry

Ve výsledku byly k naimplementování vybrány 4 filtry. Jako první bude na obraz použit medián ový filtr, který spolehlivě odstraní drobný šum. Následně bude použit Gaussův filtr s fixním kernelem zmíněným v podkapitole 7.3.3, který rozmaže ostré přechody. Poté bude vytvořen morfologický gradient, který bude odečten od obrazu a tím dojde ke zvýraznění hran. Jako poslední krok bude použito Otsu prahování.



(a) Originální obraz (b) Použité filtry

Obrázek 23: Všechny použité filtry

8 Implementace vybraných filtrů na desce FRDM-K64F

Z důvodu omezené paměti desky není možné využít knihovnu OpenCV, ani veškerá přijatá data neustále ukládat, jako v aplikaci běžící na PC. Z tohoto důvodu byly vybrané filtry naimplementovány a byla vytvořena šablona vycházející z kruhového bufferu.

8.1 Šablona Ring

V případě limitované paměti je řešením vytvoření pole omezené velikostí, které vždy obsahuje jen určitý počet nejnovějších záznamů. Zde ovšem nastává problém, pokud je pole již zaplněno a je přijat nový záznam. V takovémto případě je potřeba všechny záznamy, až na nejstarší, posunout v poli o jednu pozici, aby bylo uvolněno místo na nově přijatý záznam. Tato operace se ale stává při neustálém příjmu nových dat zbytečně náročnou, a proto byl k tomuto účelu vytvořen kruhový buffer[12].

Kruhový buffer je pole o fixní velikosti, ovšem navíc obsahuje informace pozici v poli, kde je uložen první záznam, a počtu uložených záznamů. V případě vložení nového záznamu do plně neobsazeného bufferu je pouze zvýšena hodnota odpovídající počtu záznamů. Pokud je buffer již plný, je nový záznam uložen na pozici prvního (nejstaršího) záznamu, který je tímto přepsán, a dojde k posunutí ukazatele na takto označený záznam. Kruhový buffer funguje na principu FIFO, a proto je při smazání záznamu vymazán právě záznam označený jako nejstarší, posunut ukazatel na něj a snížena hodnota označující velikost bufferu.

Ke konkrétnímu použití v této práci byla ovšem použita lehce upravená verze. Ruční mazání z bufferu není vůbec potřeba a nově obdržený záznam vždy odpovídá indexu 0. Při přijetí nového záznamu je tedy vždy pouze snížen index označující první prvek v bufferu a v případě, že buffer není plně obsazený, je zvýšena hodnota počtu prvků.

```
template <class T>
class Ring{
public:
   Ring(int maxSize) {
      this->maxSize = maxSize;
      this->arr = new T[maxSize];
      this->currentSize = this->start = 0;
   }
   void add(T val) {
      this->start = this->start == 0? this->maxSize - 1 : this->start - 1;
      if (this->currentSize < this->maxSize)
         this->currentSize++;
      this->arr[this->start] = val;
```

```
    }
    int size() {
        return this->currentSize;
    }
    int getMaxSize() {
        return this->maxSize;
    }
    T& operator[](int index) {
        return this->arr[(this->start + index) % this->maxSize];
    }
private:
    T* arr;
    int maxSize, currentSize, start;
};
```

Výpis 4: Vlastní verze kruhového bufferu

8.2 Třída ProcessedLines

Ke zpracování obrazu z kamery byla vytvořena třída nazvaná ProcessedLines, která využívá šablony Ring k ukládání historie detekovaných čar. Byly naimplementovány zvolené filtry zmíněné v podkapitole 7.6 a vlastní algoritmus k hledání okrajových čar na vyfiltrovaném obrazu.

Při vytváření objektu této třídy jsou jako parametry zadány velikosti uložené historie čar a šířek dráhy pro výpočet průměrné šířky. Řádek z kamery ke zpracování je následně předán metodě `process()`, která na něj aplikuje vybrané filtry a následně jsou nalezeny a uloženy pozice okrajových čar, a případně šířka a střed dráhy.

Je umožněno i nastavit okraje záběru, které kompenzují nevyužití celé šířky čipu kamery při použití různých objektivů. Pokud by se pracovalo s celou šířkou záběru i s nevyužitými okraji, mohlo by dojít ke zkresleným výsledkům hlavně u použití prahování a následnému vyhledávání čar. Proto probíhá veškeré filtrování a hledání pouze mezi těmito definovanými okraji.

Tato třída je použitelná jak na vývojové desce, tak přímo v prohlížeči dat z kapitoly 5.1.

8.2.1 Hledání okrajových čar

Posledním úkolem bylo najít na vyfiltrovaném obrazu okrajové čáry. Vychází se z toho, že po prahování obraz obsahuje pouze pixely s hodnotou 0 nebo 255 a právě pixely odpovídající bílé dráze mají po celé její šířce hodnotu 255. V případě rovných úseků je dráha z obou stran ohraničena černými čarami, kterým odpovídají pixely s hodnotou 0. Při průjezdu zatáčkou je toto ohraničení pouze z jedné strany a na straně druhé je obraz bílý až ke kraji záběru. Při

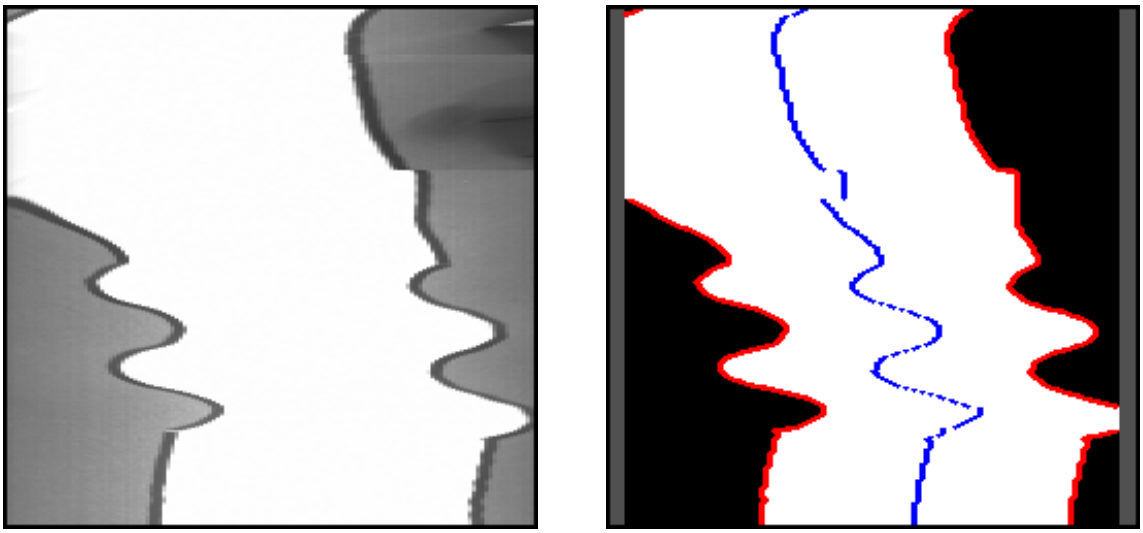
průjezdu křižovatkou je celý záběr bílý. Dále se v této části algoritmu předpokládá, že dráha je na obraze nejširší oblast bílé barvy, a právě takováto oblast se hledá.

Při hledání se tedy v cyklu projde pole hodnot reprezentující řádek z kamery a zaznamenají se okraje nejširší oblasti obsahující pouze bílé pixely. Následně se získané pozice porovnají s předchozím záznamem, a pokud je odchylka nově zjištěných pozic čar oproti předchozím menší, než určitá hodnota, která je zde nastavena na 5 pixelů, tak se takto zjištěné okrajové čáry považují za platné, a jsou uloženy jako nový záznam.

Pokud je odchylka vyšší, což může být způsobeno například nečistotou uprostřed dráhy, nebo tím, že automobil jel po kraji dráhy a byla nalezena širší oblast bílé barvy, přejde se k hledání čar od jejich pozic z předchozího záznamu. Pokud je v aktuálních datech na pozici čáry ze záznamu minulého bílá barva, začne se od této pozice směrem doleva vyhledávat první pixel s černou barvou. Je-li na této pozici černý pixel, vyhledávání bude pokračovat směrem doprava a je potřeba najít první pixel s bílou barvou. U pravé čáry se provádí totéž, ovšem v obrácených směrech. Takto nalezené čáry jsou poté uloženy jako nový záznam.

Při hledání čar může nastat situace, kdy se na jedné nebo obou stranách nebudou nacházet žádné černé pixely. Toto nastane vždy při průjezdu zatáčkou a křižovatkou. V takovémto případě je pozice levé čáry nastavena na hodnotu o 1 nižší, než levý okraj záběru, a v případě pravé čáry se bude jednat o hodnotu o 1 vyšší, než pravý okraj záběru.

Pokud jsou viditelné obě čáry, je navíc zaznamenána šířka dráhy v pixelech a přepočítána průměrná šířka, která vzniká z několika záznamů, a je určen střed dráhy, který se nachází mezi čarami. Při jednom viditelném okraji je střed dopočítán od viditelné čáry, od které je vzdálený polovinu průměrné šířky. Pokud není vidět ani jeden okraj, střed se nemění. Výsledný obraz s detekovanými čarami se nachází na obrázku 24.



(a) Obraz z kamery (b) Detekované čáry

Obrázek 24: Detekce okrajových čar

```
Lines ProcessedLines::getWidestWhiteArea(){
   Lines l;
   int16_t currentWidth = 0, maxWidth = 0, left = LEFT_BORDER - 1,
       right = RIGHT_BORDER + 1;
   l.leftLine = l.rightLine = LEFT_BORDER + RIGHT_BORDER / 2;

   for (int i = LEFT_BORDER; i <= RIGHT_BORDER; i++) {
      if (this->lastProcessedRow[i] == 0) {
         if (currentWidth > maxWidth) {
            maxWidth = currentWidth;
            l.leftLine = left;
            l.rightLine = right;
         }
         currentWidth = 0;
         left = i;
      }
      else if (this->lastProcessedRow[i] == 255) {
         right = i + 1;
         currentWidth++;
      }
   }

   if (currentWidth > maxWidth) {
      l.leftLine = left;
      l.rightLine = right;
   }
   return l;
}
```

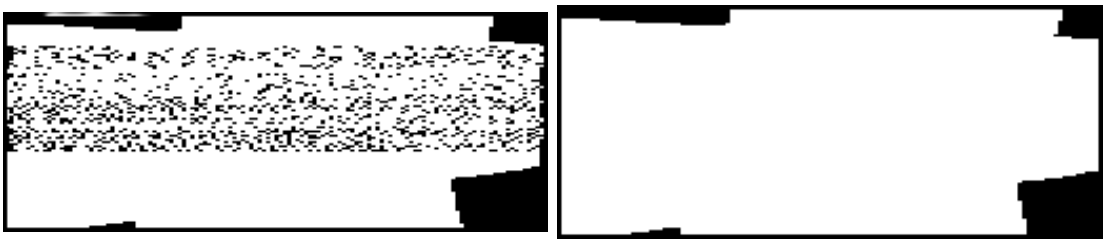
Výpis 5: Hledání nejširší bílé oblasti

8.2.2 Úprava Otsu prahování

Při implementaci Otsu prahování byl algoritmus na dvou místech lehce upraven, aby lépe vyhovoval tomuto konkrétnímu použití.

První problém, který může nastat, je při průjezdu křižovatkou. Teoreticky by měly mít všechny pixely stejnou hodnotu, jelikož je snímána celistvá bílá plocha. V praxi ovšem toto často není pravda a hodnoty pixelů se od sebe mohou o pár jednotek lišit. Proto je při vytváření histogramu obrazu zaznamenána minimální i maximální hodnota, a pokud je rozdíl těchto

hodnot nižší, než určitá hodnota, v tomto případě nastavená na 20, jsou všechny pixely obrazu nastaveny na stejnou hodnotu 255 a algoritmus je ukončen.



(a) Neupravené (b) Upravené

Obrázek 25: Porovnání neupraveného a upraveného Otsu prahování

Druhou úpravou je změna hodnoty zjištěného prahu, konkrétně její snížení. Důvodem k této úpravě je větší spolehlivost při horších světelných podmínkách. Spoléhá se na to, že okrajové čáry dráhy jsou nejtmavější částí obrazu, a proto by se jejich intenzity měly při snížení hodnoty prahu pořád nacházet pod touto novou hodnotou. Velké rozdíly intenzity po celé šířce dráhy způsobené stíny, odlesky (viz. viditelné odlesky světel na obrázku 3) a nedostatečným osvětlením ovšem po tomto kroku mohou nabývat většího množství hodnot, aniž by byly považovány za pozadí a tím pádem nastaveny na 0.

9 Závěr

Cílem této bakalářské práce bylo spolehlivě rozeznávat okrajové čáry dráhy snímané pomocí řádkové kamery. Ke kameře byl po otestování zvolen vhodný objektiv, který umožňoval následné filtrování obrazu bez přidaných problémů způsobených nesprávně zvolenými prvky snímání obrazu.

Při testování různých filtů určených k redukci šumu a detekci hran bylo zjištěno, že složitější filtry nebo velké množství zkombinovaných filtrů nemusí vždy poskytovat nejlepší výsledky.

K implementaci byly vybrány 4 filtry: mediánový filtr, Gaussův filtr, morfologický gradient a Otsu prahování. Byla vytvořena třída, kterou lze použít jak na mikropočítači, tak v prohlížeči dat.

Při použití vybraných filtrů lze tvrdit, že při rovnoměrném umělém osvětlení, které je používáno i při soutěži NXP Cup, je detekce okrajových čar na dráze spolehlivá. Při horších světelných podmínkách, například pouze při denním světle, bylo také dosaženo decentních výsledků, nelze je ovšem považovat za ideální.

Literatura

[1] LANDZO TFC Car model kit [online]. [cit. 2018-04-24]. Dostupné z http://www.landzo.com/index.php?route=product/product&product_id=95

[2] FRDM-K64F Freedom Module User's Guide [online]. [cit. 2018-04-24]. Dostupné z https://www.nxp.com/docs/en/user-guide/FRDMK64FUG.pdf

[3] Freescale Cup Shield for the Freedom KL25Z [online]. [cit. 2018-04-24]. Dostupné z https://community.nxp.com/docs/DOC-93914

[4] The NXP Cup EMEA [online]. [cit. 2018-04-24]. Dostupné z https://community.nxp.com/groups/tfc-emea

[5] Doxygen [online]. [cit. 2018-04-24]. Dostupné z http://www.stack.nl/~dimitri/doxygen/

[6] Sokety a C++ [online]. [cit. 2018-04-24]. Dostupné z http://www.builder.cz/rubriky/c/c--/sokety-a-c--156186cz

[7] OpenCV [online]. [cit. 2018-04-24]. Dostupné z https://docs.opencv.org/3.4.1/

[8] Obrazové filtry [online]. [cit. 2018-04-24]. Dostupné z https://is.mendelu.cz/eknihovna/opory/zobraz_cast.pl?cast=18431

[9] Digital filters [online]. [cit. 2018-04-24]. Dostupné z http://homepages.inf.ed.ac.uk/rbf/HIPR2/filtops.htm

[10] Dilation, erosion, and the morphological gradient [online]. [cit. 2018-04-24]. Dostupné z https://blogs.mathworks.com/steve/2006/09/25/dilation-erosion-and-the-morphological-gradient/

[11] Otsu Thresholding [online]. [cit. 2018-04-24]. Dostupné z http://www.labbookpages.co.uk/software/imgProc/otsuThreshold.html

[12] Circular buffer [online]. [cit. 2018-04-24]. Dostupné z https://en.wikipedia.org/wiki/Circular_buffer

A Soubor tfc.h

```
/**
@file tfc.h
*/

#ifndef __TFC_H
#define __TFC_H

#include <stdint.h>

#define TFC_VERSION "20171213"

#define __TFC_EMBEDED__ 1

/** Resolution of the camera. */
#define TFC_CAMERA_LINE_LENGTH  128

/** Specifies the integer range for PWM, as in <-::TFC_PWM_MINMAX, ::
    TFC_PWM_MINMAX>, which is used in this application's methods
TFC::getMotorPWM_i() and TFC::setMotorPWM_i().
*/
#define TFC_PWM_MINMAX        1000

/** Specifies the integer range for servo, as in <-::TFC_SERVO_MINMAX, ::
    TFC_SERVO_MINMAX>, which is used in this application's methods
TFC::getServo_i() and TFC::setServo_i().
*/
#define TFC_SERVO_MINMAX      1000

/** Maximum value of analog sample obtained from ADC by the method TFC::
    ReadADC(). */
#define TFC_ADC_MAXVAL        0x0FFF

/** Specifies the integer ranges for analog values, as in <-::
    TFC_ANDATA_MINMAX, ::TFC_ANDATA_MINMAX> or <0, ::TFC_ANDATA_MINMAX>,
    which are used in this application's methods
TFC::ReadPot_i(), TFC::ReadFB_i() and TFC::ReadBatteryVoltage_i().
*/
```

```
#define TFC_ANDATA_MINMAX     1000

/** Default center position of servos - pulse width in microseconds. */
#define TFC_SERVO_DEFAULT_CENTER 1500

/** Specifies the default offset from ::TFC_SERVO_DEFAULT_CENTER, which
    corresponds to fully turned wheels to either side. */
#define TFC_SERVO_DEFAULT_MAX_LR 200

/** Specifies the maximal offset from the servos' center, which can be used
    in calibration by method TFC::setServoCalibration(). */
#define TFC_SERVO_MAX_LR      400

/** Specifies the default integer range for PWM, as in <-::
    TFC_PWM_DEFAULT_MAX, ::TFC_PWM_DEFAULT_MAX>, which can be changed in the
    method
TFC::setPWMMax() up to <-::TFC_PWM_MINMAX, ::TFC_PWM_MINMAX>.
*/
#define TFC_PWM_DEFAULT_MAX      200

/** Maximal allowed -/+ PWM duty cycle for the underlying HW. */
#define HW_TFC_PWM_MAX        500

/** Command used in a packet for sending/receiving data. */
#define CMD_DATA           1

/** Command used in a packet for calibration. */
#define CMD_SETTING           2

/** Command used in a packet for controlling features of the board or
    receiving data from them. */
#define CMD_CONTROL           3

/** Packet header. */
#define STX                0x2
/** Packet footer. */
#define ETX                0x3
```

```
/** @brief Order of analog data in an array. These values are used as a
    parameter for a method TFC::ReadADC(). */
enum tfc_andata_chnl_enum
{
   anPOT_1, ///< Potentiometer 1.
   anPOT_2, ///< Potentiometer 2.
   anFB_A,    ///< Feedback A from the H-Bridge.
   anFB_B,    ///< Feedback B from the H-Bridge.
   anBAT,     ///< Battery voltage.
   anLast     ///< Last member used as a size of an array.
};


/**
@brief Empty data packet

This packet contains only necessary parametres, no actual data. \n
When using variation of this packet to send/receive data, the data should be
     put in a place of ::data. \n
There can be array of N entries of the data structure (::tfc_data_s, ::
    tfc_setting_s or ::tfc_control_s) in one packet.
*/
struct tfc_protocol_empty_s
{
   uint8_t stx;  ///< Each packet must start with a ::STX byte.
   uint16_t length;///< Length of the whole packet.
   uint8_t cmd;  ///< Type of command, can be either ::CMD_DATA, ::
       CMD_SETTING or ::CMD_CONTROL.
   char[0] data; ///< Placeholder for the actual data.
   uint8_t etx;  ///< Each packet must end with an ::ETX byte.
};

/**
@brief Structure containing analog data and data from camera.

This data structure contains data corresponding to a certain timestamp. Is
    usually sent in packet (::tfc_protocol_empty_s) with ::CMD_DATA. \n
*/
struct tfc_data_s
```

```
{
   uint32_t timestamp;              ///< Number of the sample.
   uint16_t adc[ anLast ];          ///< All analog data specified in ::
       tfc_andata_chnl_enum.
   uint8_t    DIP_sw;                 ///< Values of DIP switches.
   uint8_t    push_sw;              ///< Values of push buttons.
   uint16_t image[ TFC_CAMERA_LINE_LENGTH ];///< One line from the camera.
   uint32_t _padding;               ///< Padding bytes.
};

/**
@brief Structure for calibration of the individual features.

This data structure is usually sent in packet (::tfc_protocol_empty_s) with
    ::CMD_SETTING at the start of the application. \n
*/
struct tfc_setting_s
{
   uint16_t servo_center[ 2 ];      ///< Center of the servos.
   uint16_t servo_max_lr[ 2 ];      ///< Offset from the center of the servos
        corresponding to their maximal rotation.
   uint16_t pwm_max;           ///< Maximal PWM for motors.
   uint16_t _padding;             ///< Padding bytes.
};

/**
@brief Structure for controlling individual features or receiving their data
    .

This data structure is usually sent in packet (::tfc_protocol_empty_s) with
    ::CMD_CONTROL. \n
*/
struct tfc_control_s
{
   uint8_t    leds;              ///< Values of the 4 LEDs in the form of a
       low nibble.
   uint8_t  pwm_onoff;           ///< On/off value of the motors [0,1].
   uint8_t  servo_onoff;         ///< On/off value of the servos [0,1].
   uint8_t  _padding1;           ///< Padding byte.
```

```
    int16_t  pwm[ 2 ];            ///< PWM values of the motors <-::
        TFC_PWM_MINMAX, ::TFC_PWM_MINMAX>.
    int16_t  servo_pos[ 2 ];        ///< Positions of the servos <-::
        TFC_SERVO_MAX_LR, ::TFC_SERVO_MAX_LR>.
};


/**
* @brief Main class for controlling all the features.
*/
class TFC
{
public:
    TFC() { InitAll(); }

    /**
    * Initialize all setting variables to it's default values.
    */
    void InitAll();

    /** @name LEDs
    These methods are used to control LEDs.\n
    */
    ///@{

    /**
    * Set a single LED.
    * @param led Number of the LED <0,3>.
    * @param val On/off value of the LED.
    */
    void setLED( uint32_t led, uint32_t val );

    /**
    * Set multiple leds.
    * @param leds 4 leds in the form of a low nibble (the lowest 4 bits in
        the variable), where each bit value [0,1] turns on/off each LED.
    */
    void setLEDs( uint32_t leds );
```

```
/**
* Set LEDs corresponding to the battery level.
* @param bat_level Value in range 0-4.
*/
void setBatteryLEDLevel( uint32_t bat_level );
///@}


/** @name Buttons and DIP switch
These methods are used to obtain values from the DIP switch or the
    buttons.\n
*/
///@{

/**
* Get value of the DIP switch.
* @return Value of the DIP switch.
*/
uint32_t getDIPSwitch();

/**
* Get value of a push button.
* @param channel Corresponds to a button, 0 - first button, any other
    value - second button.
* @return Value of the button [0,1].
*/
uint32_t getPushButton( uint32_t channel );
///@}


/**
* Get analog value directly from the ADC.
* @param andata_chnl Type of a analog value to get, can be any value from
     ::tfc_andata_chnl_enum .
* @return Value corresponding to the selected channel in the range <0, ::
    TFC_ADC_MAXVAL>.
*/

uint32_t ReadADC( uint32_t andata_chnl );
```

```
/** @name Potentiometers
These methods are used to obtain values from the potentiometers.\n
@code
The potentiometers have their own value range,
   +-------------------------|-------------------------+
which looks like this:                                  0
                     "Middle"                  MAX

When the value is obtained from the ADC,
   +-------------------------|-------------------------+
the range is translated to this:                        0
                 TFC_ADC_MAXVAL/2          TFC_ADC_MAXVAL

The ReadPot_i() method translates the ADC values
   +-------------------------|-------------------------+
to this range:                                  -TFC_ANDATA_MINMAX
               0               TFC_ANDATA_MINMAX

And the ReadPot_f() method further translates
   +-------------------------|-------------------------+
the values to this range:                             -1.0
                       0                    1.0
@endcode
*/
///@{

/**
* Get integer value of a potentionmeter.

* @param Channel 0 - first potentiometer, any other value - second
   potentiometer.
* @return Value in range <-::TFC_ANDATA_MINMAX , ::TFC_ANDATA_MINMAX >.
*/
int32_t ReadPot_i( uint32_t Channel );

/**
* Get float value of a potentionmeter. \n
```

```
* @param Channel 0 - first potentiometer, any other value - second
    potentiometer.
* @return Value in range <-1.0, 1.0>.
*/
float ReadPot_f( uint32_t Channel );
///@}


/** @name Feedback
These methods are used to obtain feedback values.\n

@code
When the value of the feedback is obtained
    +-------------------------|-------------------------+
from the ADC, the range looks like this:                    0
                  TFC_ADC_MAXVAL/2        TFC_ADC_MAXVAL

The ReadFB_i() method translates the ADC values
    +-------------------------|-------------------------+
to this range:                                             0
                TFC_ANDATA_MINMAX/2     TFC_ANDATA_MINMAX

And the ReadPot_f() method further translates
    +-------------------------|-------------------------+
the values to the actual electrical current:               0
                      2.8125                  5.625
@endcode
*/
///@{

/**
* Get integer feedback value.
* @param Channel 0 - first FB channel, any other value - second FB
    channel.
* @return Value in range <0, ::TFC_ANDATA_MINMAX>.
*/
uint32_t ReadFB_i( uint32_t Channel );

/**
```

```
* Get feedback value as current.
* @param Channel 0 - first FB channel, any other value - second FB
    channel.
* @return Curremt in Amperes.
*/
float ReadFB_f( uint32_t Channel );
///@}


/** @name Battery
These methods are used to obtain battery values.\n

@code
When the voltage on the battery is obtained
    +------------------------|------------------------+
from the ADC, the range looks like this:                    0
                  TFC_ADC_MAXVAL/2        TFC_ADC_MAXVAL

The ReadBatteryVoltage_i() method translates
    +------------------------|------------------------+
the ADC values to this range:                              0
                TFC_ANDATA_MINMAX/2     TFC_ANDATA_MINMAX

And the ReadBatteryVoltage_f() method further translates
    +------------------------|------------------------+
the values to the actual voltage on the battery:           0
                       9.405                   18.81
@endcode
*/
///@{

/**
* Get integer representation of voltage on battery.
* @return Value in range <0, ::TFC_ANDATA_MINMAX>.
*/
uint32_t ReadBatteryVoltage_i();

/**
* Get voltage on battery.
```

```
* @return Value in Volts.
*/
float ReadBatteryVoltage_f();
///@}


/** @name Camera
These methods are used to obtain data from a camera.\n
*/
///@{

/**
* Check whether camera is ready or not.
* @param channel 0 - first cammera, any other value - possible second
    camera.
* @return 0 - is not ready, 1 - camera is ready.
*/
uint32_t ImageReady( uint32_t channel );

/**
* Get image (line) from a camera.
* @param channel 0 - first cammera, any other value - possible second
    camera.
* @param img Pointer to an uint16_t array representing the image.
* @param length Size of the array, should be ::TFC_CAMERA_LINE_LENGTH.
*/

void getImage( uint32_t channel, uint16_t *img, uint32_t length );
///@}


/** @name Servos
These methods are used to control servos.\n

@code
The servos are controlled by PWM, specifically
by the pulse width duration int microseconds,
    +------------------------|------------------------+
```

```
so their value range looks by default like this:  TFC_SERVO_DEFAULT_CENTER
     TFC_SERVO_DEFAULT_CENTER TFC_SERVO_DEFAULT_CENTER
(or it can be calibrated by setServoCalibration()) -
    TFC_SERVO_DEFAULT_MAX_LR                        +TFC_SERVO_DEFAULT_MAX_LR

In the methods getServo_i() and setetServo_i()
    +-------------------------|-------------------------+
the range is translated to this:                         -TFC_SERVO_MINMAX
                0                TFC_SERVO_MINMAX

And the setServo_f() method further translates
    +-------------------------|-------------------------+
the values to this range:                                    -1.0
                   0                1.0
@endcode

*/
///@{

/**
* Servo calibration.
* @param channel Channel 0 - first servo, any other value - second servo.
* @param center Pulse width in microseconds corresponding to the center
    position of the servo.
* @param max_lr Offset from center to either side corresponding to the
    maximal rotation, in the form of a pulse width in microseconds.
*/

void setServoCalibration( uint32_t channel, uint32_t center, uint32_t
    max_lr );

/**
* Enable or disable both servos.
* @param onoff Disable or enable servos [0,1].
*/
void ServoOnOff( uint32_t onoff );

/**
* Set servo position.
```

```
* @param channel Channel 0 - first servo, any other value - second servo.
* @param position Position in range <-::TFC_SERVO_MINMAX , ::
    TFC_SERVO_MINMAX>.
*/

void setServo_i( uint32_t channel, int32_t position );

/**
* Get current position of servo.
* @param channel Channel 0 - first servo, any other value - second servo.
* @return Value in range <- ::TFC_SERVO_MINMAX , ::TFC_SERVO_MINMAX>.
*/
int32_t getServo_i( uint32_t channel );

/**
* Set servo position.
* @param channel Channel 0 - first servo, any other value - second servo.
* @param position Value in range <-1.0,1.0>.
*/
void setServo_f( uint32_t channel, float position );
///@}


/** @name Motors
These methods are used to control motors.\n

@code
The motors are controlled by PWM, so their
duty cycle range looks like this:
    +------------------------|------------------------+
(negative values correspond to reverse directon):          -100%
                        0%                      100%

In the methods getMotorPWM_i() and setMotorPWM_i()
    +------------------------|------------------------+
the range is translated to this:                        -TFC_PWM_MINMAX
                    0                TFC_PWM_MINMAX
```

```
And the setMotorPWM_f() method further translates
    +-------------------------|-------------------------+
the values to this range:                                -1.0
                         0                       1.0
@endcode

*/
///@{

/**
* Set maximal PWM duty cycle for the motors. This limits the motors to
    the given value, so when the maximal value is set by this method for
    example to 300 and then the method
* TFC::setMotorPWM_i() is called with values greater than that, the
    motors will only run at this maximal value, i.e. 300.
* @param max Pulse width in microseconds in the range <0, ::
    TFC_PWM_MINMAX>.
*/
void setPWMMax( uint32_t max );

/**
* Enable or disable both motors.
* @param onoff Disable or enable motors [0,1].
*/
void MotorPWMOnOff( uint32_t onoff );

/**
* Set PWM for both motors in range <-::TFC_PWM_MINMAX , ::TFC_PWM_MINMAX
    >.
* If the values are greater than the maximum set by the method TFC::
    setPWMMax(), the motors will only run at that set maximum.
* @param pwm_a Value for first motor.
* @param pwm_b Value for second motor.
*/
void setMotorPWM_i( int32_t pwm_a, int32_t pwm_b );

/**
* Get current PWM integer value of a motor.
```

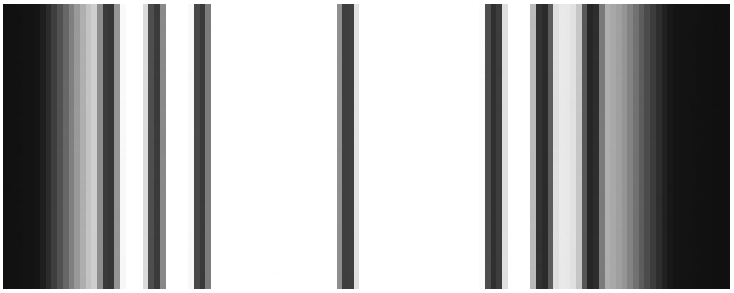
```
* @param channel_ab Channel 0 - first motor, any other value - second
    motor.
* @return Value in range <-::TFC_PWM_MINMAX , ::TFC_PWM_MINMAX>.
*/
int32_t getMotorPWM_i( uint32_t channel_ab );

/**
* Set PWM for both motors in range <-1.0, 1.0>.
* @param pwm_a Value for first motor.
* @param pwm_b Value for second motor.
*/
void setMotorPWM_f( float pwm_a, float pwm_b );
///@}

/** Local calibration of the servos and the motors. */
tfc_setting_s m_setting;

/** Local object used to store current values of leds, servos and motors.
     */
tfc_control_s m_control;

#ifdef __TFC_EMBEDED__

/** @name Remote Control
These methods are used to control and obtain data from the RC pins.
*/
///@{

/**
* Initialize SPEED0 and SPEED1 pins used for RC.
*/
void InitRC();

/**
* Enable or disable pulses detection on SPEED0 and SPEED1 pins.
* @param onoff 0 for disable 1 for enable.
*/
void RCOnOff( uint32_t onoff );
```

```
    /**
    * Get last pulse width from the RC.
    * @param channel Specify from which channel to get the pulse width [0,1].
    * @return Last pulse width in microseconds.
    */
    uint32_t getRCPulse( uint32_t channel );
    ///@}


    #else

    void setData( s_data *data );
    s_setting *getSetting();
    s_control *getControl();

protected:

    tfc_data_s m_data;
    int m_data_ready;

#endif // __TFC_EMBEDED__

};

#endif // __TFC_H
```

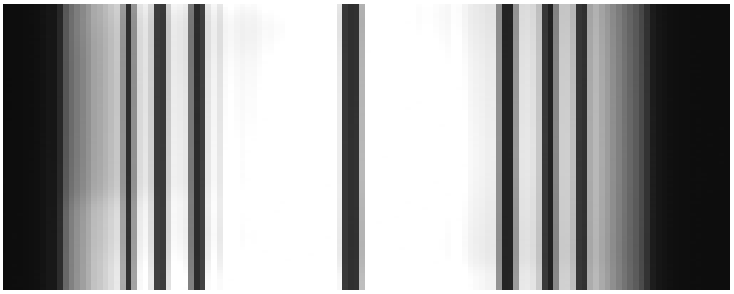
Výpis 6: tfc.h

B Záběry testovaných objektivů

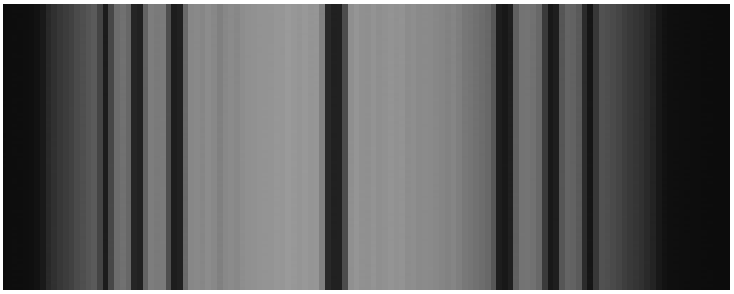
B.1 Kamera č. 1



Obrázek 26: Kamera č. 1 a objektiv č. 1

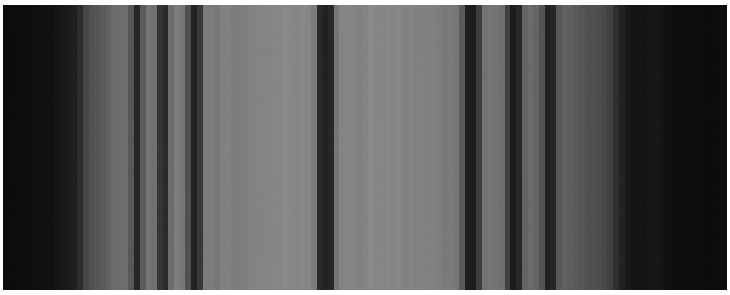


Obrázek 27: Kamera č. 1 a objektiv č. 2

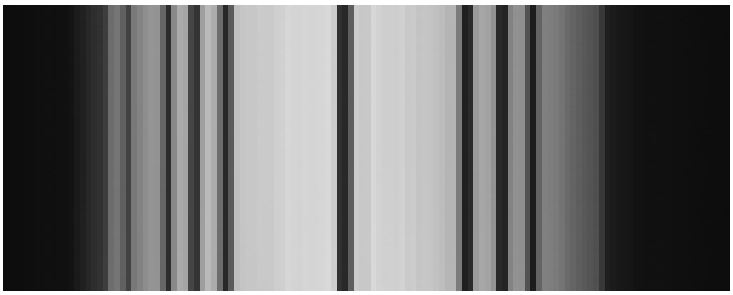


Obrázek 28: Kamera č. 1 a objektiv č. 3

Obrázek 29: Kamera č. 1 a objektiv č. 4



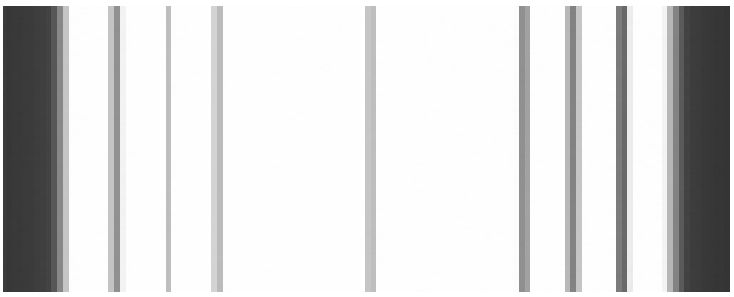
Obrázek 30: Kamera č. 1 a objektiv č. 5



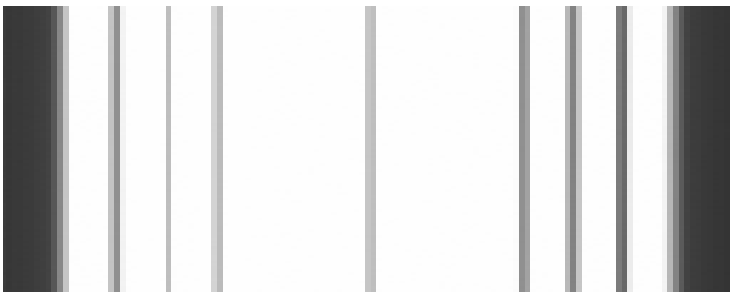
Obrázek 31: Kamera č. 1 a objektiv č. 6

Obrázek 32: Kamera č. 1 a objektiv č. 7

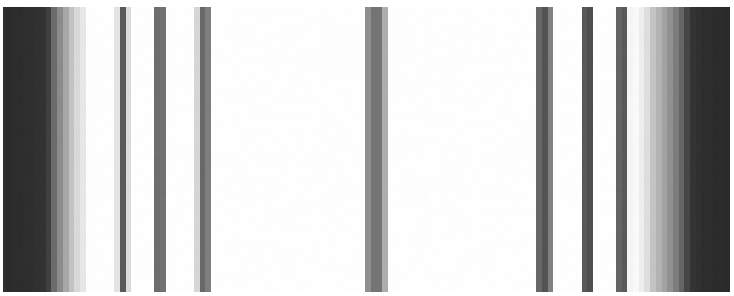
B.2 Kamera č. 2



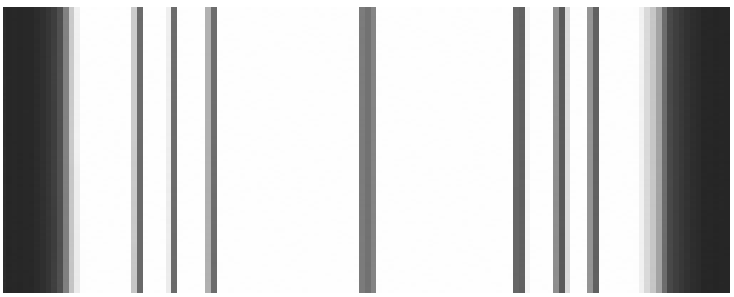
Obrázek 33: Kamera č. 2 a objektiv č. 1



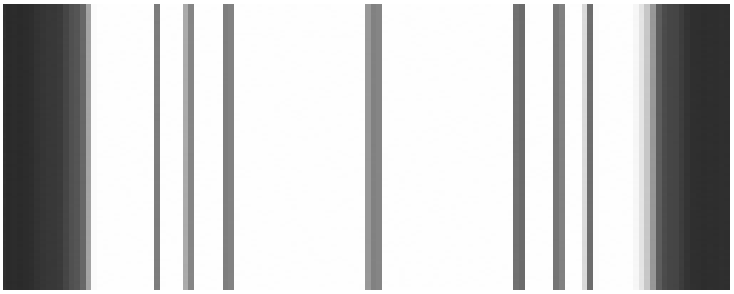
Obrázek 34: Kamera č. 2 a objektiv č. 2



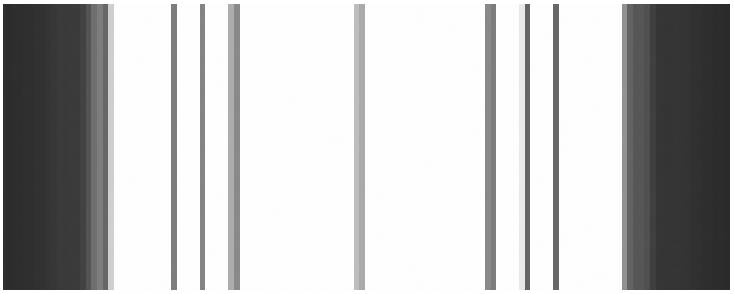
Obrázek 35: Kamera č. 2 a objektiv č. 3



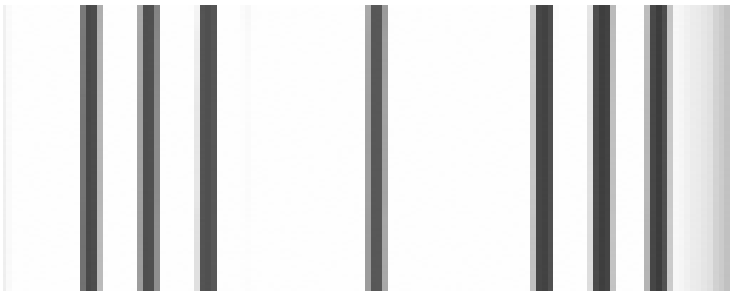
Obrázek 36: Kamera č. 2 a objektiv č. 4



Obrázek 37: Kamera č. 2 a objektiv č. 5



Obrázek 38: Kamera č. 2 a objektiv č. 6



Obrázek 39: Kamera č. 2 a objektiv č. 7

C Obsah přiloženého CD

- Prázdný projekt k použití na mikrokontroléru desky FRDM-K64F
- Vygenerovaná dokumentace k souboru tfc.h
- Aplikace k prohlížení a odesílání dat
- Záznamy z testování objektivů a z průjezdů tratí